EEE3096S Practical 1B Report

Chris Whittaker WHTCHR013

Emily Slettevold SLTEMI002

I. Introduction

The practical aims to analyse and measure the performance of an STM32F0 processor using a common benchmark of running a Mandelbrot function. The STM32 will run the Mandelbrot function to calculate the checksum and execution time of the code in an embedded system. The accuracy of the STM32's checksum will be evaluated by comparing it to the checksum from the Python reference code that is run on a laptop. Two methods of doing the Mandelbrot calculation are utilized. The first makes use of fixed point arithmetic and the other, floating point arithmetic, which uses doubles. The testing will be done using various image dimensions.

II. Methodology

1. Open the main.c file provided.
2. Define required (global) variables: imagine_dimensions, checksum, start_time, end_time and elapsed_time.
3. Complete the function using the provided python code or pseudocode: uint64_t calculate_mandelbrot_fixed_point_arithmetic(int width, int height, int max_iterations). This method should make use of fixed point arithmetic to ensure that no rounding is done through the use of integer types. Code the "uint64_t calculate_mandelbrot_double(int width, int height, int max_iterations)" method, making use of doubles instead of integers.
4. Assign HAL_GetTick() to start_time and end_time variables before and after the calculate_mandelbrot_fixed_point_arithmetic function call.
5. The difference between start_time and end_time is used to calculate the elapsed_time value.
6. Set imag_dim (image dimensions) to 128.
7. Press the debug button and select 'live expression'.
8. Add variables 'checksum' and 'elapsed_time' to the pane.
9. Press the 'resume' button to run the code in debug mode.
10. Wait for the outputs and record the variable values.
11. Repeat steps 6 to 10 for image_dimensions 160, 192, 224, 256.
12. Repeat steps 6 to 11 using "uint64_t calculate_mandelbrot_double(in

t width, int height, int max_iterations)”

13. Run the python code and ensure that the recorded values for checksum using both methods are within 1% of the python-produced values.

III. Results and Discussion

The practical yielded satisfactory results as the recorded checksum results have a less than 1% tolerance in comparison to the python-produced results. As seen by the images attached in the appendix below, the double function produced the same values as the python code values, and the fixed point arithmetic values were less than 1 % off from these values.

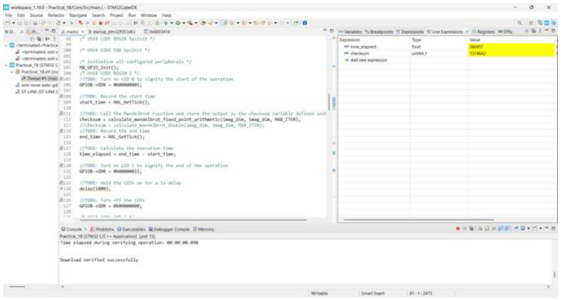
IV. Conclusion

The Mandelbrot set benchmark was successfully implemented to analyse the capabilities of the STM32F0 compared to a laptop. Methods using both fixed point arithmetic and doubles were used. Both methods yielded results which were in a 1% tolerance or less than the reference checksums produced by the provided Python code for square images with dimensions of 128, 160, 192, 224 and 256 pixels. Thus, the practical achieved the expected results.

However, the code can be improved by making use of bit shifting instead of multiplication and division, which are computationally expensive and take a long time to complete. This would significantly reduce the run time of each test if implemented within the Mandelbrot functions (specifically the for loops).

V. AI Clause

The use of AI proved to be useful during this assignment. The use of fixed point arithmetic is required for a portion of the code. Although, an outline of the workings for this is included in Appendix 6, an error occurred due to the use unsigned 64-bit integers when calculating the checksum in the Mandelbrot function. The LLM was able to identity the error. It revealed that the error was to do with subtractions giving incorrect values with unsigned integers. Hence, AI provided an efficient debugging tool with useful explanations.



```
/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */
//TODO: Turn on LED 0 to signify the start of the operation
GPIOB->ODR = 0b00000001;

//TODO: Record the start time
start_time = HAL_GetTick();

//TODO: Call the Mandelbrot Function and store the output in the checksum variable defined init
checksum = calculate_mandelbrot_fixed_point_arithmetic(image_dim, image_dim, MAX_ITER);
//checksum = calculate_mandelbrot_double(image_dim, image_dim, MAX_ITER);
//TODO: Record the end time
end_time = HAL_GetTick();

//TODO: Calculate the execution time
time_elapsed = end_time - start_time;

//TODO: Turn on LED 1 to signify the end of the operation
GPIOB->ODR = 0b00000011;

//TODO: Hold the LEDs on for a 1s delay
delay(1000);

//TODO: Turn off the LEDs
GPIOB->ODR = 0b00000000;
```

Expression	Type	Value
time_elapsed	float	386857
checksum	uint64_t	1314642

```
Time elapsed during verifying operation: 00:00:00.090

Download verified successfully
```

Image size of 224x224 pixels

Appendix:

Run's with the Mandelbrot function using fixed point arithmetic:

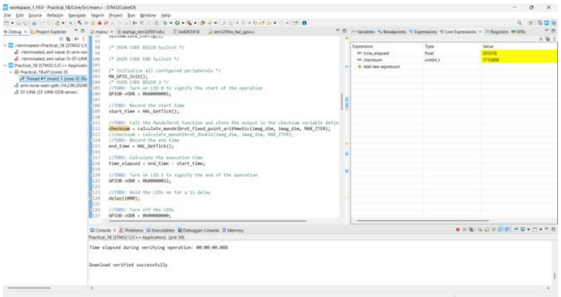


Image size of 256x256 pixels

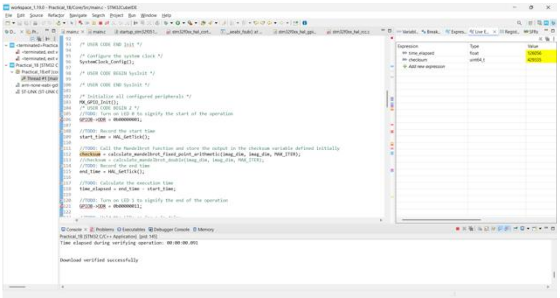


Image size of 128x128 pixels

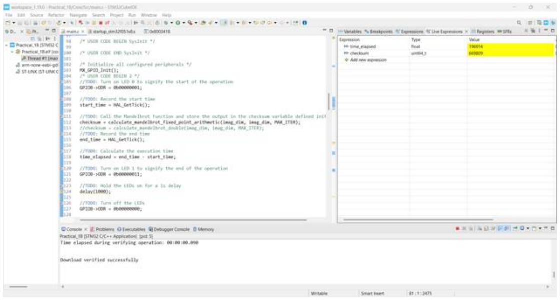


Image size of 160x160 pixels

Run's with the Mandelbrot function using doubles:

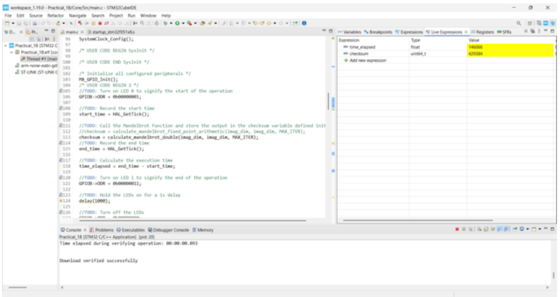


Image size of 128x128 pixels

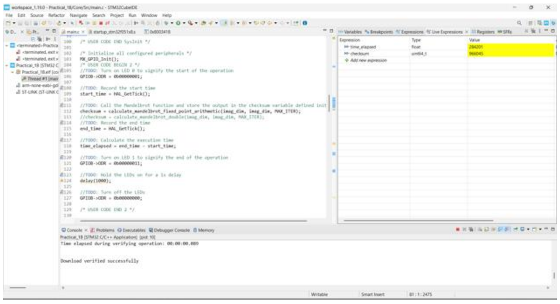


Image size of 192x192 pixels

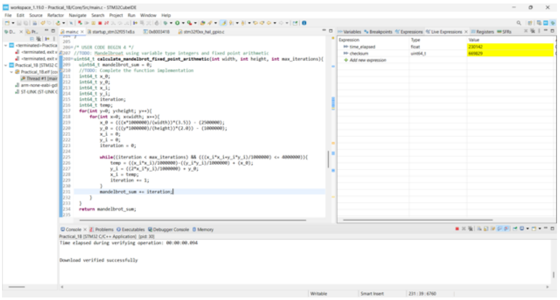
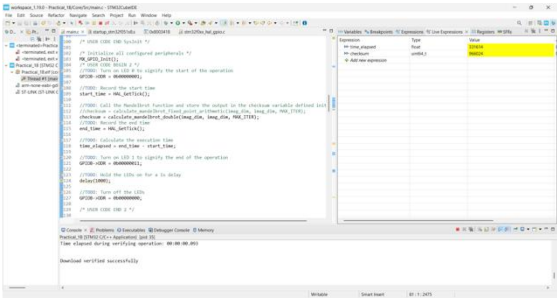


Image size of 160x160 pixels

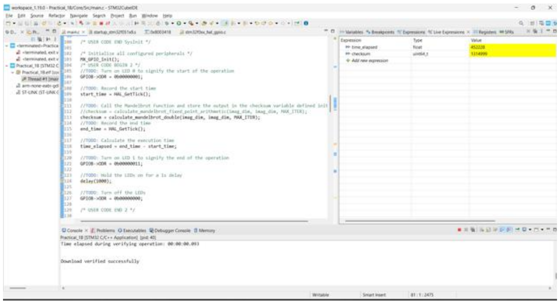


```
/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */
//TODO: Turn on LED 0 to signify the start of the operation
GPIOB->ODR = 0b00000001;

//TODO: Record the start time
start_time = HAL_GetTick();

//TODO: Call the Mandelbrot Function and store the output in the checksum variable defined init
//checksum = calculate_mandelbrot_fixed_point_arithmetic(imag_dim, imag_dim, MAX_ITER);
checksum = calculate_mandelbrot_double(imag_dim, imag_dim, MAX_ITER);
//TODO: Record the end time
end_time = HAL_GetTick();

//TODO: Calculate the execution time
time_elapsed = end_time - start_time;

//TODO: Turn on LED 1 to signify the end of the operation
GPIOB->ODR = 0b00000011;

//TODO: Hold the LEDs on for a 1s delay
delay(1000);

//TODO: Turn off the LEDs
GPIOB->ODR = 0b00000000;

/* USER CODE END 2 */
```

Expression	Type	Value
time_elapsed	float	331614
checksum	uint64_t	866024

```
Practical_1B [STM32 C/C++ Application] [pid: 35]
Time elapsed during verifying operation: 00:00:00.093

Download verified successfully
```

Image size of 192x192 pixels

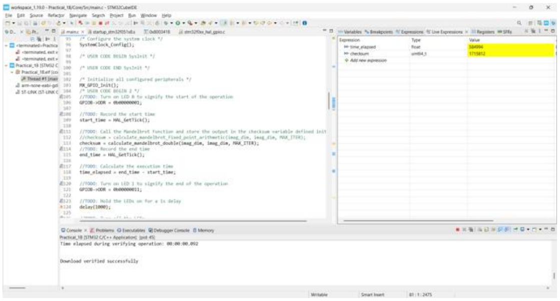


```
/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */
//TODO: Turn on LED 0 to signify the start of the operation
GPIOB->ODR = 0b00000001;

//TODO: Record the start time
start_time = HAL_GetTick();

//TODO: Call the Mandelbrot Function and store the output in the checksum variable defined init
//checksum = calculate_mandelbrot_fixed_point_arithmetic(imag_dim, imag_dim, MAX_ITER);
checksum = calculate_mandelbrot_double(imag_dim, imag_dim, MAX_ITER);
//TODO: Record the end time
end_time = HAL_GetTick();

//TODO: Calculate the execution time
time_elapsed = end_time - start_time;

//TODO: Turn on LED 1 to signify the end of the operation
GPIOB->ODR = 0b00000011;

//TODO: Hold the LEDs on for a 1s delay
delay(1000);

//TODO: Turn off the LEDs
GPIOB->ODR = 0b00000000;

/* USER CODE END 2 */
```

Expression	Type	Value
time_elapsed	float	452228
checksum	uint64_t	1314999

```
Practical_1B [STM32 C/C++ Application] [pid: 40]
Time elapsed during verifying operation: 00:00:00.093

Download verified successfully
```

Image size of 224x224 pixels



```
/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */
//TODO: Turn on LED 0 to signify the start of the operation
GPIOB->ODR = 0b00000001;

//TODO: Record the start time
start_time = HAL_GetTick();

//TODO: Call the Mandelbrot Function and store the output in the checksum variable defined init
//checksum = calculate_mandelbrot_fixed_point_arithmetic(imag_dim, imag_dim, MAX_ITER);
checksum = calculate_mandelbrot_double(imag_dim, imag_dim, MAX_ITER);
//TODO: Record the end time
end_time = HAL_GetTick();

//TODO: Calculate the execution time
time_elapsed = end_time - start_time;

//TODO: Turn on LED 1 to signify the end of the operation
GPIOB->ODR = 0b00000011;

//TODO: Hold the LEDs on for a 1s delay
delay(1000);
```

Expression	Type	Value
time_elapsed	float	584994
checksum	uint64_t	1715812

```
Practical_1B [STM32 C/C++ Application] [pid: 45]
Time elapsed during verifying operation: 00:00:00.092

Download verified successfully
```

Image size of 256x256 pixels

Code for Prac 1B:

```
/* USER CODE BEGIN Header */
/**
*************************************
*************************************
****
* @file : main.c
* @brief : Main program body
*************************************
*************************************
****
* @attention
*
* Copyright (c) 2025
STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms
that can be found in the LICENSE file
* in the root directory of this
software component.
* If no LICENSE file comes with this
software, it is provided AS-IS.
*
*************************************
*************************************
****
*/
/* USER CODE END Header */
/* Includes -------------------------
-------------------------------------
---*/
#include "main.h"

/* Private includes -----------------
-------------------------------------
---*/
/* USER CODE BEGIN Includes */
#include <stdint.h>
#include "stm32f0xx.h"
/* USER CODE END Includes */

/* Private typedef ------------------
-------------------------------------
---*/
/* USER CODE BEGIN PTD */
#define MAX_ITER 100
/* USER CODE END PTD */

/* Private define -------------------
-------------------------------------
---*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro --------------------
-------------------------------------
---*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ----------------
-------------------------------------
---*/

/* USER CODE BEGIN PV */
```

```c
//TODO: Define and initialise the global varibales required
/*
start_time
end_time
execution_time
checksum: should be uint64_t
initial width and height maybe or you might opt for an array??
*/
uint64_t imag_dim = 256;
uint64_t checksum;
float start_time;
float end_time;
float time_elapsed;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */
uint64_t calculate_mandelbrot_fixed_point_arithmetic(int width, int height, int max_iterations);
uint64_t calculate_mandelbrot_double(int width, int height, int max_iterations);


/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
* @brief The application entry point.
* @retval int
*/
int main(void)
{
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/* MCU Configuration--------------------------------------------------------*/

/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */
//TODO: Turn on LED 0 to signify the start of the operation
GPIOB->ODR = 0b00000001;

//TODO: Record the start time
start_time = HAL_GetTick();
//TODO: Call the Mandelbrot Function and store the output in the checksum variable defined initially
checksum = calculate_mandelbrot_fixed_point_arithmetic(imag_dim, imag_dim, MAX_ITER);
//checksum = calculate_mandelbrot_double(imag_dim, imag_dim, MAX_ITER);
//TODO: Record the end time
end_time = HAL_GetTick();

//TODO: Calculate the execution time
time_elapsed = end_time - start_time;
```

```
//TODO: Turn on LED 1 to signify the
end of the operation
GPIOB->ODR = 0b00000011;

//TODO: Hold the LEDs on for a 1s delay
delay(1000);

//TODO: Turn off the LEDs
GPIOB->ODR = 0b00000000;

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}

/**
* @brief System Clock Configuration
* @retval None
*/
void SystemClock_Config(void)
{
RCC_OscInitTypeDef RCC_OscInitStruct =
{0};
RCC_ClkInitTypeDef RCC_ClkInitStruct =
{0};

/** Initializes the RCC Oscillators
according to the specified parameters
* in the RCC_OscInitTypeDef structure.
*/
RCC_OscInitStruct.OscillatorType     =
RCC_OSCILLATORTYPE_HSI;
RCC_OscInitStruct.HSIState           =
RCC_HSI_ON;
RCC_OscInitStruct.HSICalibrationValue
= RCC_HSICALIBRATION_DEFAULT;
RCC_OscInitStruct.PLL.PLLState       =
RCC_PLL_NONE;
if
(HAL_RCC_OscConfig(&RCC_OscInitStruct
) != HAL_OK)
{
Error_Handler();
}

/** Initializes the CPU, AHB and APB
buses clocks
*/
RCC_ClkInitStruct.ClockType          =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSC
LK
|RCC_CLOCKTYPE_PCLK1;
RCC_ClkInitStruct.SYSCLKSource       =
RCC_SYSCLKSOURCE_HSI;
RCC_ClkInitStruct.AHBCLKDivider      =
RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider     =
RCC_HCLK_DIV1;

if
(HAL_RCC_ClockConfig(&RCC_ClkInitStru
ct, FLASH_LATENCY_0) != HAL_OK)
{
Error_Handler();
}
}

/**
* @brief GPIO Initialization Function
* @param None
* @retval None
*/
static void MX_GPIO_Init(void)
{
GPIO_InitTypeDef   GPIO_InitStruct   =
{0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

/* GPIO Ports Clock Enable */
__HAL_RCC_GPIOB_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB,
GPIO_PIN_0|GPIO_PIN_1,
GPIO_PIN_RESET);

/*Configure GPIO pins : PB0 PB1 */
GPIO_InitStruct.Pin                  =
GPIO_PIN_0|GPIO_PIN_1;
```

```
GPIO_InitStruct.Mode =
GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed =
GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB,
&GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
//TODO: Mandelbroat using variable
type integers and fixed point
arithmetic
uint64_t
calculate_mandelbrot_fixed_point_arit
hmetic(int width, int height, int
max_iterations){
uint64_t mandelbrot_sum = 0;
//TODO: Complete the function
implementation
int64_t x_0;
int64_t y_0;
int64_t x_i;
int64_t y_i;
int64_t iteration;
int64_t temp;
for(int y=0; y<height; y++){
 for(int x=0; x<width; x++){
 x_0 = (((x*1000000)/(width))*(3.5)) -
(2500000);
 y_0 = (((y*1000000)/(height))*(2.0))
- (1000000);
 x_i = 0;
 y_i = 0;
 iteration = 0;

 while((iteration < max_iterations) &&
(((x_i*x_i+y_i*y_i)/1000000) <=
4000000)){
 temp = ((x_i*x_i)/1000000)-
((y_i*y_i)/1000000) + (x_0);
 y_i = ((2*x_i*y_i)/1000000) + y_0;
 x_i = temp;
 iteration += 1;
 }
 mandelbrot_sum += iteration;
 }
}
return mandelbrot_sum;

}

//TODO: Mandelbroat using variable
type double
uint64_t
calculate_mandelbrot_double(int width,
int height, int max_iterations){
uint64_t mandelbrot_sum = 0;
//TODO: Complete the function
implementation
double x_0;
double y_0;
double x_i;
double y_i;
double iteration;
double temp;
for(double y=0; y<height; y++){
for(double x=0; x<width; x++){
 x_0 = ((x/width)*3.5) - 2.5;
 y_0 = ((y/height)*2.0) - 1.0;
 x_i = 0;
 y_i = 0;
iteration = 0;

 while((iteration < max_iterations) &&
((x_i*x_i) + (y_i*y_i) <= 4)){
temp = (x_i*x_i)-(y_i*y_i) + x_0;
y_i = (2*x_i*y_i) + y_0;
x_i = temp;
iteration += 1;
 }
 mandelbrot_sum += iteration;
}
}
return mandelbrot_sum;
}

/* USER CODE END 4 */

/**
* @brief This function is executed in
case of error occurrence.
* @retval None
*/
void Error_Handler(void)
{
```

```
/* USER CODE BEGIN Error_Handler_Debug
*/
/* User can add his own implementation
to report the HAL error return state
*/
__disable_irq();
while (1)
{
}
/* USER CODE END Error_Handler_Debug
*/
}

/**
* Basic delay function
*/
void delay(uint32_t delay_time){
uint32_t i = (delay_time*8000)/8;
uint32_t i_in = i/1000;
for(volatile uint32_t j=0; j<1000;
j++){
 for(volatile uint32_t m=0; m<(i_in);
m++){

 }
}
}

#ifdef USE_FULL_ASSERT
/**
* @brief Reports the name of the source
file and the source line number
* where the assert_param error has
occurred.
* @param file: pointer to the source
file name
* @param line: assert_param error line
source number
* @retval None
*/
void assert_failed(uint8_t *file,
uint32_t line)
{
/* USER CODE BEGIN 6 */
/* User can add his own implementation
to report the file name and line
number,
ex: printf("Wrong parameters value:
file %s on line %d\r\n", file, line)
*/
/* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```